A LIST APART

Illustration by Kevin Cornell

Fluid Grids

by **Ethan Marcotte** · March 03, 2009

Published in CSS, HTML, Graphic Design, Layout & Grids, Responsive Design

Early last year, I worked on the redesign of a rather content-heavy website *(http://www.w3.org/QA/2008/06/about_the_love_w3org_redesign.html)*. Design requirements were fairly light: the client asked us to keep the organization's existing logo and to improve the dense typography and increase legibility. So, early on in the design process, we spent a sizable amount of time planning a well-defined grid for a library of content modules.

81 comm

Over the past few years, this sort of thinking has become more common. Thanks to the advocacy of Mark Boulton *(http://www.markboulton.co.uk/)*, Khoi Vinh *(http://subtraction.com/2007/03/18/oh-yeeaahh)*, and others, we've seen a resurgence of interest in the typographic grid, and how to use it on the web. And frankly, the idea's been a smash hit: a *(http://960.gs/)* million *(http://www.blueprintcss.org/)* CSS *(http://developer.yahoo.com/yui/grids/)* frameworks *(http://csswizardry.com/typogridphy/)* have bloomed, with sundry *(http://www.puidokas.com/portfolio/gridfox/)* tools *(http://developer.yahoo.com/yui/grids/builder/)* to complement them, each built to make grid-based design even more accessible to the average designer. And why not? After a few minutes of griddy thinking, the benefits become clear: designers gain a rational, structured framework for organizing content and users gain well-organized, legible sites.

However, our client had one last, heart-stopping requirement: the design had to be fluid and resize with the browser window. Normally, this would cause me to rejoice both noisily and embarrassingly. Fluid layouts are an undervalued commodity in web design. They put control of our designs firmly in the hands of our users *(http://www.alistapart.com/articles/dao/)* and their browsing habits. They've also utterly failed to seize the imagination of web designers.

Minimum screen resolution: a little white lie

Instead of exploring the benefits of flexible web design, we rely on a little white lie: "minimum screen resolution." These three words contain a powerful magic, under the cover of which we churn out fixed-width layout after fixed-width layout, perhaps revisiting a design every few years to "bump up" the width once it's judged safe enough to do so. "Minimum screen resolution" lets us design for a contrived subset of users who see our design as god and Photoshop intended. These users always browse with a maximized 1024—768 window, and are never running, say, an OLPC laptop *(http://laptop.org/en/laptop/)*, or looking at the web with a monitor that's more than four years old. If a user doesn't meet the requirements of "minimum screen resolution," well, then, it's the scrollbar for them, isn't it?

Of course, when I was coding the site, I didn't have the luxury of writing a diatribe on the evils of fixed-width design. Instead, I was left with a sobering fact: while we'd designed a rather complex grid to serve the client's content needs, the client—and by extension, the client's *users*—was asking for a fluid layout. As almost all of the grid-based designs I could list off at that time were rigidly fixed-width, I was left with a prickly question: how *do* you create a fluid grid?

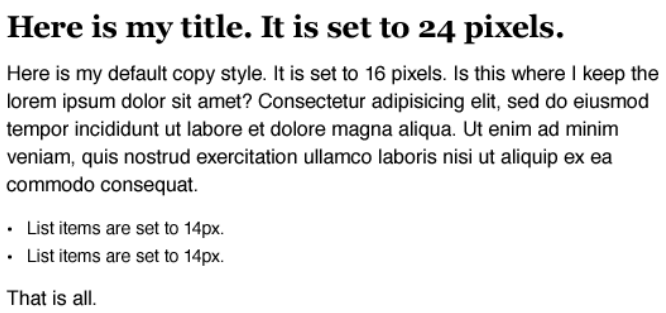
As it turns out, it's simply a matter of context.

Do I really have to thank IE for this?

Faced with an insurmountable problem, I did what I do best: avoid it altogether. Temporarily putting aside the question of *how* to get a grid to behave in a non-fixed layout, I coded the stuff I knew: styles first for color and backgrounds, and then for setting the type.

You may already know about Internet Explorer's well-documented problem with resizing fonts set in pixels —or rather, its utter refusal to do so *(http://www.simplebits.com/notebook/2008/03/25/ie8.html)*. Set a paragraph in 16px Georgia, and no matter how much the user tries to increase or decrease the size of the text, it remains at 16px in IE. IE7 and onward do allow the user to scale the entire page, but simple resizing of px-based fonts is still largely *verboten* in Internet Explorer. So to give our users the most flexibility, we standards-savvy designers have usually opted to sidestep the pixel entirely, and have taken to sizing type with relative units, be they keywords *(http://www.w3.org/TR/CSS21/fonts.html#value-def-absolute-size)*, percentages *(http://www.w3.org/TR/CSS21/syndata.html#value-def-percentage)*, or my personal favorite, ems *(http://www.alistapart.com/articles/howtosizetextincss/)*.

If you've ever worked with relative units *(http://www.w3.org/TR/CSS21/syndata.html#length-units)* such as the em, you know that it's all about context: in other words, the actual size of an element's em is computed relative to the `font-size` of its parent element. For example, let's say we're working from the following design comp:



An example of some basic text sized using pixels.

Nothing fancy: some paragraphs set in 16px Helvetica, an unordered list that's been slightly downsized to 14px, and an `h1` at the top in 24px Georgia. Sexy, no?

What's doubly sexy is that one simple rule allows us to get most of this in place *(/d/fluidgrids/examples/type/initial.html)*:

```
body {
    font: normal 100% Helvetica, Arial, sans-serif;
}
```

With a `font-size` of `100%`, all the elements in our page are sized relative to the browser's default type size, which in most cases is 16px. And thanks to the browser's default stylesheet, the `h1` is big, bold, and beautiful—but still in Helvetica, and much too large. So while it'd be easy enough to slap on a `font-family` to fix the header's Helvetica problem, how do we size the text to 24 pixels? Or accurately reduce the size of that list?

With ems, it's easily done. We take the target value for each element's `font-size` in pixels and divide it by the `font-size` of its container (that is, its context). We're left with the desired `font-size`, expressed in relative, em-friendly terms. Or to put it more succinctly:

```
target ÷ context = result
```

If we assume the `body`'s default type size to be 16px, we can plug each desired `font-size` value into this formula. So to properly match our header to the comp, we divide the target value (24px) by the `font-size` of its container (16px):

```
24 ÷ 16 = 1.5
```

So the header is 1.5 times the default body size, or 1.5em, which we can plug directly into our stylesheet.

```
h1 {
    font-family: Georgia, serif;
    font-size: 1.5em;        /* 24px / 16px = 1.5em */
```

```
}
```

To size the list to the em-equivalent of 14px, we can use the same formula. Assuming again that the `body`'s `font-size` is roughly 16px, we simply divide that target by the context:

```
14 ÷ 16 = 0.875
```

And we're left with a value of 0.875em, which we can again drop into our CSS.

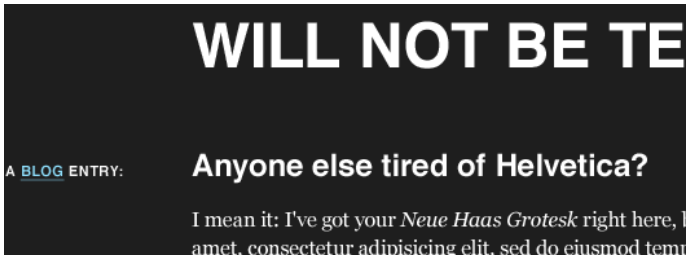
```
ul {
    font-size: 0.875em;      /* 14px / 16px = 0.875em
*/
}
```

With those two rules, our sample page *(/d/fluidgrids/examples/type/sizes.html)* is looking a lot closer to the comp, and will be practically pixel-perfect after some slight cleanup *(/d/fluidgrids/examples/type/polished.html)*. All with the help of our `target ÷ context = result` formula.

So after a few hours spent cleaning up relative type styling for our client, I realized I'd stumbled upon the answer. If we could treat font sizes not as pixels, but as *proportions* measured against their container, we could do the same with the different elements draped across our grid.

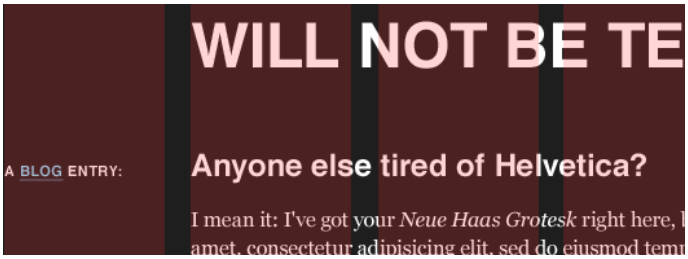
After all, it's not "The Golden Pixel"

As before, let's start with a fairly ~~unsexy~~ straightforward layout:



Our basic page layout. (/d/fluidgrids/img/comp-full.gif)

Sure, our "design" *(/d/fluidgrids/img/comp-full.gif)* is pretty modest. But those simple styles are draped over a well-defined grid *(/d/fluidgrids/img/comp-grid.gif)*: namely, seven columns of 124px each, separated by 20px-wide gutters, all of which totals up to a width of 988px. But hey, let's forget about those nasty pixels. Proportions are the new black, right? Let's get fluid, baby.



Our basic page, with the grid overlaid upon it. (/d/fluidgrids/img/comp-grid.gif)

To start, let's treat our comp like any other, fixed or fluid: before we start coding, let's look at the design, and assess the different content areas *(/d/fluidgrids/img/comp-areas.gif)*. Thankfully, it's a pretty short inventory.

Defining our different content areas. (/d/fluidgrids/img/comp-areas.gif)

On the highest level *(/d/fluidgrids/img/comp-areas.gif)*, we've got a title at the top, a content area that spreads across six columns, and some contextual information in the leftmost column. From this diagram, we can flesh out some skeleton markup that keys into our content inventory, both structurally and semantically:

```
<div id="page">
    <h1>The Ratio Revolution Will Not Be Televised</h1>
    <div class="entry">
        <h2>Anyone else tired of Helvetica?</h2>
        <h3 class="info">A <a href="#">Blog</a> Entry:</h3>
        <div class="content">
            <div class="main">
                <p>Main content goes here. Lorem ipsum etc., etc.
</p>
            </div><!-- /end .content -->
            <div class="meta">
                <p>Posted on etc., etc.</p>
            </div><!-- /end .meta -->
        </div><!-- /end .main -->
    </div><!-- /end .entry -->
</div><!-- /end #page -->
```

And with some type rules applied, we've got a respectable-looking starting point *(/d/fluidgrids/examples/grid/initial.html)*. However, the `#page` container doesn't have any constraints on it, so our content will simply reflow to match the width of the browser window. Let's try to rein in those long line lengths a bit:

```
#page {
    margin: 40px auto;
    padding: 0 1em;
    max-width: 61.75em;      /* 988px / 16px = 61.75em
*/
}
```

We've used margins and padding to ventilate our design *(/d/fluidgrids/examples/grid/bounded.html)* a bit, and establish a gutter between it and the window edges. But in the last line of our rule, we're using a variant of our `font-size` formula to define the maximum width of our design. By dividing the comp's width of 988px by our base `font-size` of 16px, we can set a `max-width` in ems to approximate the pixel-based width from our mockup, which will prevent the page from exceeding our ideal of 988px. But since we've used ems to set this upper limit, the `max-width` will scale up as the user increases her browser's text size—a nifty little trick that even works in older versions of Internet Explorer, if a small CSS patch *(http://www.cameronmoll.com/archives/000892.html)* is applied.

So with our design properly cordoned off, let's begin working on each element in our design inventory, beginning with the page's title. In the comp, it spans five columns and their four gutters, with a total width of 700px. It's also removed from the left-hand edge of the page by one column/gutter pair, making for a nice 144px offset. And if we were working in a fixed-width design, our job would be pretty straightforward:

```
h1 {
    margin-left: 144px;
    width: 700px;
}
```

Since we're working in a fluid context, though, fixed measurements don't quite cut it. And as I was working on relative font sizing, that's when it hit me: every aspect of the grid—and the elements laid upon it—can be expressed as a proportion relative to its container. In other words, as in our type resizing exercise, we're looking not just at the desired size of the element, but also at **the relationship of that size to the element's container**. This will allow us to convert our design's pixel-based widths into percentages, and keep the proportions of our grid intact as it resizes.

In short, we'll have a fluid grid.

Everything old is new again

So, how do we begin?

```
target ÷ context = result
```

That's right: it's the return of our trusty type formula. We can use the same proportional analysis to transform pixel-based column widths into percentage-based, *flexible* measurements. So we're working from a target value of 700px for the page's title—but it's contained within a designed width of 988px.

Converting our pixel-based title to percentages.

As a result, we simply divide 700px (the target) by 988px (the context) like so:

```
700 ÷ 988 = 0.7085
```

And there it is: 0.7085 translates into 70.85%, a `width` we can drop directly into our stylesheet:

```
h1 {
    width: 70.85%;          /* 700px / 988px = 0.7085
*/
}
```

Can we do the same with our target margin of 144px? Oh, I do so love a leading question:

```
144 ÷ 988 = 0.14575
```

Once again, we can take that 0.14575, or 14.575%, and add that directly to our style rule as a value for the title's `margin-left`:

```
h1 {
    margin-left: 14.575%;    /* 144px / 988px = 0.14575
*/
    width: 70.85%;          /* 700px / 988px = 0.7085
*/
}
```

And voilà *(/d/fluidgrids/examples/grid/header.html)*. By measuring the title's margin and width in relation to its container, we've successfully translated the ratios from our grid into CSS-friendly percentages. The title's proportions will always remain intact, even as it reflows to fit the size of the browser window.

We can even perform the same simple division to wrap up the layout for the entry itself, sized at 844px in our comp, with some 124px-wide marginalia to the left of it. For the entry:

```
844 ÷ 988 = 0.85425
```

And for the informational column:

```
124 ÷ 988 = 0.12551
```

These two quick divisions net us some percentages that we can drop into our stylesheet, fleshing out our layout even more:

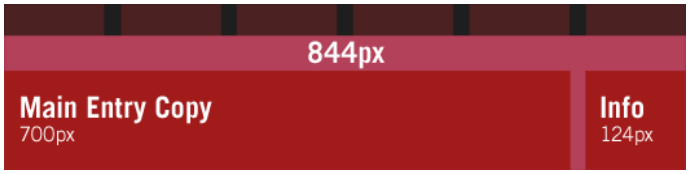
```
.entry h2,
.entry .content {
    float: right;
    width: 85.425%;          /* 844px / 988px = 0.85425
*/
}.entry .info {
    float: left;
    width: 12.551%;          /* 124px / 988px = 0.12551
*/
}
```

And with that, our fluid grid shapes up a bit further *(/d/fluidgrids/examples/grid/entry.html)*.

Changing the context

So far we've got the big content areas sorted, but we've yet to touch the inner area. Currently, the blog entry's main copy and its contextual info occupy the full width of the entry, and are stacked on top of each other. But in our initial comp *(/d/fluidgrids/img/comp-full.gif)*, the main copy inside the blog entry only spanned five columns, with the ancillary info slotted neatly into the rightmost column.

Sharp readers will have noticed that, as it's currently designed, the entry's body is the same width as the page's title (700px), and the marginalia is the same width as the leftmost column we styled earlier (124px). So while we're working with some dimensions we've previously calculated, we can't reuse the same formulas: the context has changed.



Since we're working inside a new container, we need to use its width as our context.

Whereas before we were calculating percentages relative to the 988px-wide `#page`, we're currently working within `.entry .content`, which is noticeably smaller. So as a result, we need to redefine our context, and work off the designed width of `.entry .content` as our reference point. So to define the percentage-based width of the main copy, we take its designed width of 700px, and divide it by 844px:

```
700 ÷ 844 = 0.82938
```

And for our 124px-wide column on the right, we can use the same reference point:

```
124 ÷ 844 = 0.14692
```

We can now take each of these measurements, and plug them into our CSS:

```
.entry .main {
    float: left;
    width: 82.938%;          /* 700px / 844px = 0.82938
*/
}.entry .meta {
    float: right;
    width: 14.692%;          /* 124px / 844px = 0.14692
*/
}
```

And with that we've finished our work, our fluid grid complete *(/d/fluidgrids/examples/grid/final.html)*.

A note on rounding

As you may guess from the lack of CSS patches above, I've had very few cross-browser issues with this technique. I would highly recommend John Resig's excellent article on *Sub-Pixel Problems in CSS (http://ejohn.org/blog/sub-pixel-problems-in-css/)*. It explains how different browsers handle percentage-based widths, and the mechanics by which they reconcile sub-pixel measurements.

As John explains in his article, if modern browsers are presented with four 25%-wide elements within a 50px-wide container, they can't actually render the elements at 12.5px; instead, most will round the columns down or up as best fits the layout. Internet Explorer, as it happens, will simply round all of those

sub-pixel values up, which breaks layouts.

If you're working with sufficiently generous margins in your grid, this shouldn't be an issue. But if IE causes undue wrapping with your percentage-based columns, reducing the *target* value by one pixel can help. So if, for example, our left-hand marginalia was too wide for IE (Internet Explorer), you might change your calculation from:

```
124 ÷ 988 = 0.12551
```

to a lower target of 123px:

```
123 ÷ 988 = 0.12449
```

Plug that `width` of 12.449% into your IE-specific stylesheet, and your layout woes should clear right up.

A grid for all seasons

The above is, of course, a starting point: there are myriad other challenges that face the liquid web designer, most of which arise when you introduce fixed content (such as images, Flash, and so forth) into a fluid framework. I've been experimenting with a few possible solutions on my blog *(http://unstoppablerobotninja.com/)*, but other, better workarounds are still out there.

And finally, I don't pretend that design is easy, whether it's fixed or fluid. But given all that we've achieved over the past few years—moving past tables, evangelizing standards in our companies and in our shared industry, demanding better standards of our browsers and our peers—I do wish we'd bend some of that ingenuity to break out of our reliance on "minimum screen resolution." After all, our users' browsing habits aren't as fixed as our comps would suggest. I hope the promise of fluid grids has fired your imagination, and I'm excited to see how you improve on the technique. Our users will be, too.

Additional reading

As you may have gathered from my introductory ~~crazed rant~~ digression, two passions of mine are f
design and, more recently, the importance of a well-considered grid. Both of these have been fuele
following, though this isn't an exhaustive list:

- John Allsopp, *A Dao of Web Design (http://www.alistapart.com/articles/dao/)*
- Mark Boulton, *Feeling your way around grids (http://www.markboulton.co.uk/)*
- David Emery, *More Width (http://de-online.co.uk/2006/11/27/more-width)*
- Molly Holzschlag, *Thinking Outside the Grid (http://www.alistapart.com/articles/outsidethe*
- Jeremy Keith, *The unpushed envelope (http://adactio.com/journal/1149)*
- Jeffrey Zeldman, *Rules-based design (http://www.zeldman.com/daily/0403b.shtml#ap3003*

And finally, at the end of a talk I gave last August on designing for fluid grids, someone pointed ou
Fluid 960 Grid System *(http://www.designinfluences.com/fluid960gs/)*. If you're using a public
framework such as 960 Grid System *(http://960.gs/)* already, the fluid "port" might be of interest

d

Mo' Pixels
Mobile device
higher and high
and laptops are
well. T

About the Author

Ethan Marcotte

Ethan Marcotte *(http://ethanmarcotte.com/)* is an independent web designer who cares deeply about beautiful design, elegant code, and the intersection of the two. Over the years, his clientele has included New York Magazine, the Sundance Film Festival, The Boston Globe, and People Magazine. Ethan can be found on Twitter *(http://twitter.com/beep)* or on his long-neglected blog *(http://unstoppablerobotninja.com/)*, and his most recent book is Responsive Web Design *(http://www.abookapart.com/products/responsive-web-design)*.

MORE FROM THIS AUTHOR

Coming June 16: Pattern Language *(http://alistapart.com/blog/post/style-guide-event)*, The Only Constant is Change: A Q&A with Ethan Marcotte *(http://alistapart.com/blog/post/responsive-web-design-second-ed)*, Blue Beanie Day 14: Toque 'em if You've Got 'em *(http://alistapart.com/blog/post/blue-beanie-day-14-toque-em-if-youve-got-em)*, 10 Years Ago in ALA: Pocket Sized Design *(http://alistapart.com/blog/post/10-years-ago-in-ala-pocket-sized-design)*

ISSN 1534-0295